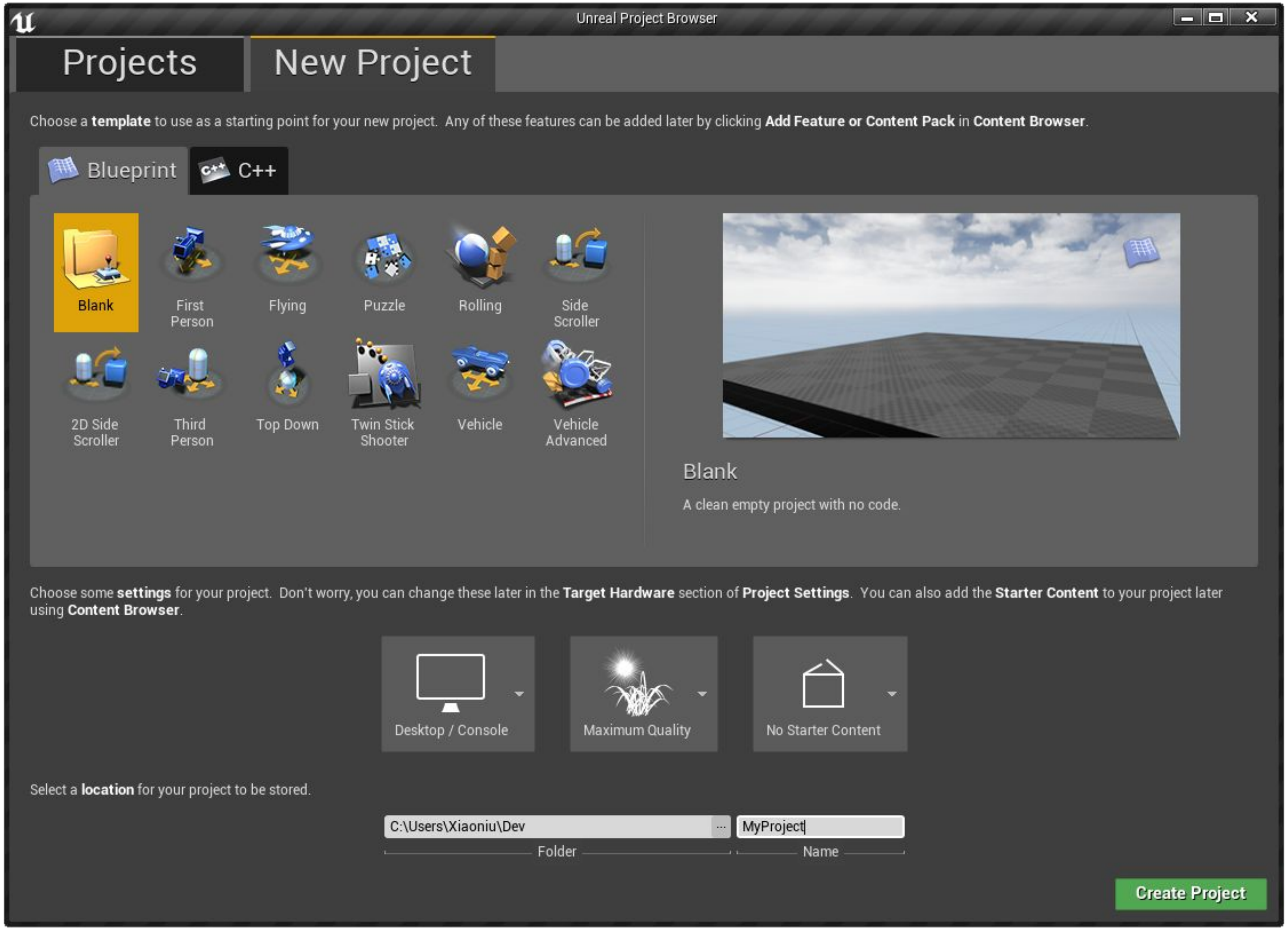
Content

Overview

PerceptionNeuron plugin provides a function to import motion capture data issued by Noitom motion capture equipment PerceptionNeuron to UnrealEngine to drive the movement of appropriate character. PerceptionNeuron can use motion capture data from two kinds of data source. One is issued by Axis Neuron, includes real-time motion capture data and pre-recorded motion capture data played by Axis Neuron. The other is BVH asset imported to the engine previously. Now we’re going to introduce detailed usage of PerceptionNeuron plugin.

Create a test project

We start with creating a test project. For the sake of simplicity, we establish a blank project "MyProject" based on the Blueprint:



Choose a **template** to use as a starting point for your new project. Any of these features can be added later by clicking **Add Feature or Content Pack** in **Content Browser**.

Blank

A clean empty project with no code.

Choose some **settings** for your project. Don't worry, you can change these later in the **Target Hardware** section of **Project Settings**. You can also add the **Starter Content** to your project later using **Content Browser**.

Desktop / Console

Maximum Quality

No Starter Content

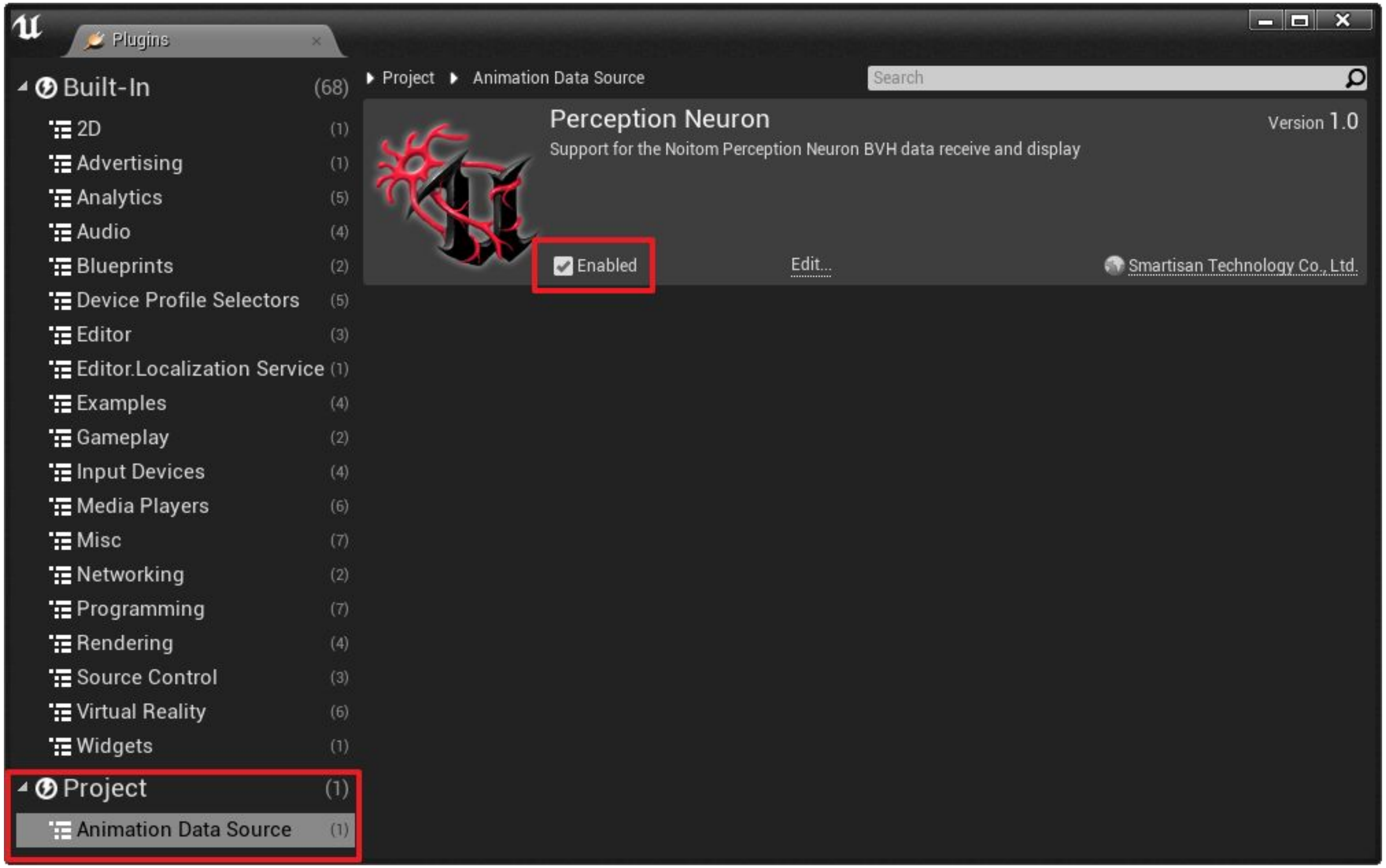
Select a **location** for your project to be stored.

Folder: C:\Users\Xiaoniu\Dev

Name: MyProject

Close the editor after creating the test project, then create **Plugins** folder in the root directory of the project, copy PerceptionNeuron plugin to the

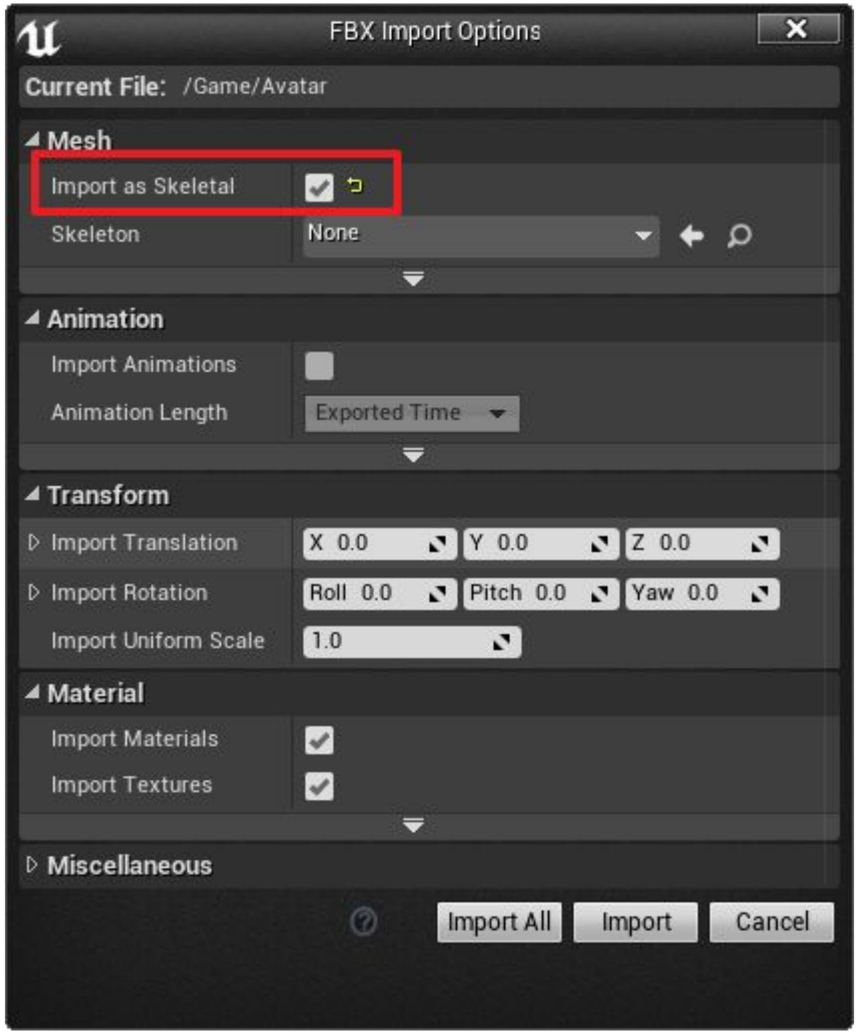
Plugins directory. Restart the editor and open the test project, open **Editor->Plugins** to check whether PerceptionNeuron plugin is successfully installed and has started with editor as below:



Perception Neuron
Support for the Noitom Perception Neuron BVH data receive and display
Version 1.0
✓ Enabled

Create AnimationBlueprint

Now we can import the pre-made model with skeleton. We import a FBX file and must select **”Import as Skeletal”** in import configuration:


FBX Import Options

Current File: /Game/Avatar

Mesh
Import as Skeletal ✓
Skeleton None

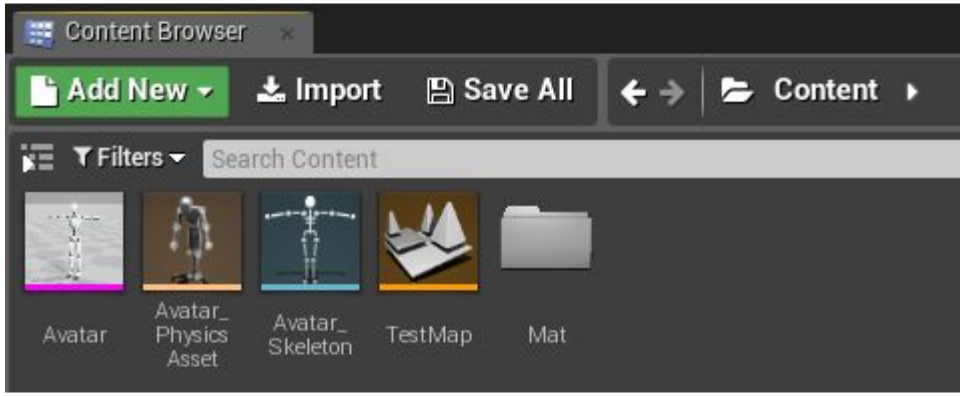
Animation
Import Animations
Animation Length Exported Time

Transform
Import Translation X 0.0 Y 0.0 Z 0.0
Import Rotation Roll 0.0 Pitch 0.0 Yaw 0.0
Import Uniform Scale 1.0

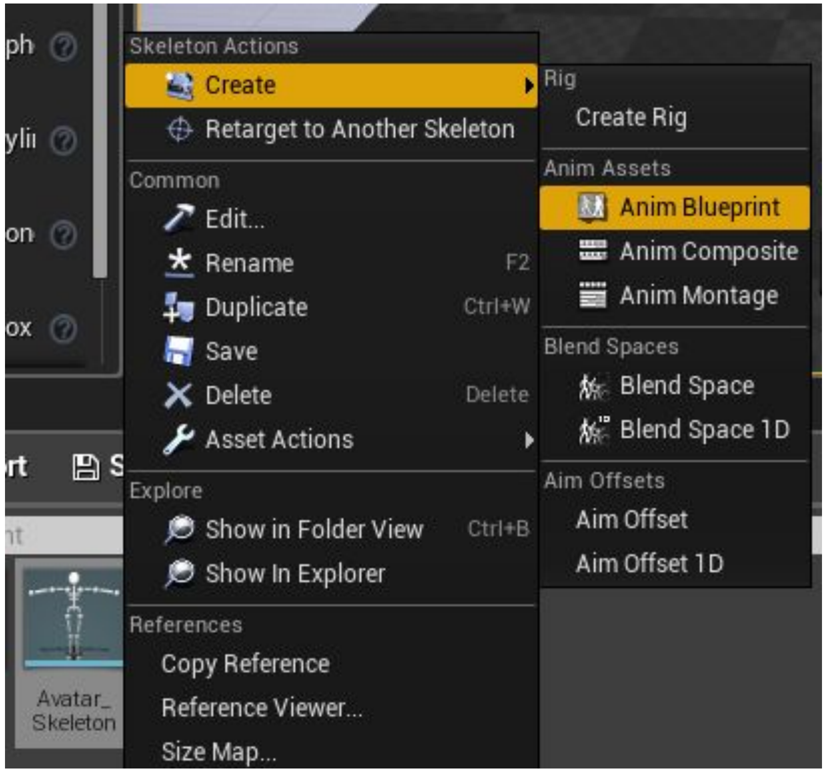
Material
Import Materials ✓
Import Textures ✓

Miscellaneous

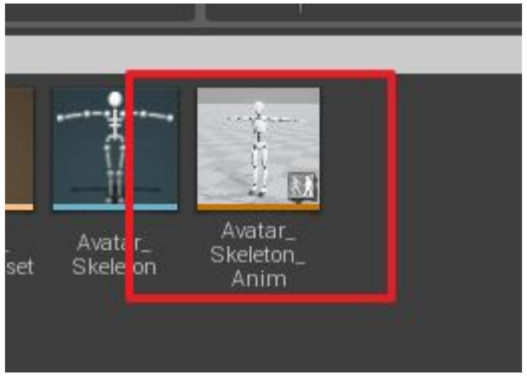
Now you can see that **Avatar** and **Avatar_Skeleton** have been import to engine successfully in Content Browser:



Then we create Animation Blueprint based on **Avatar_Skeleton**. Right-click on **Avatar_Skeleton** to create Animation Blueprint:



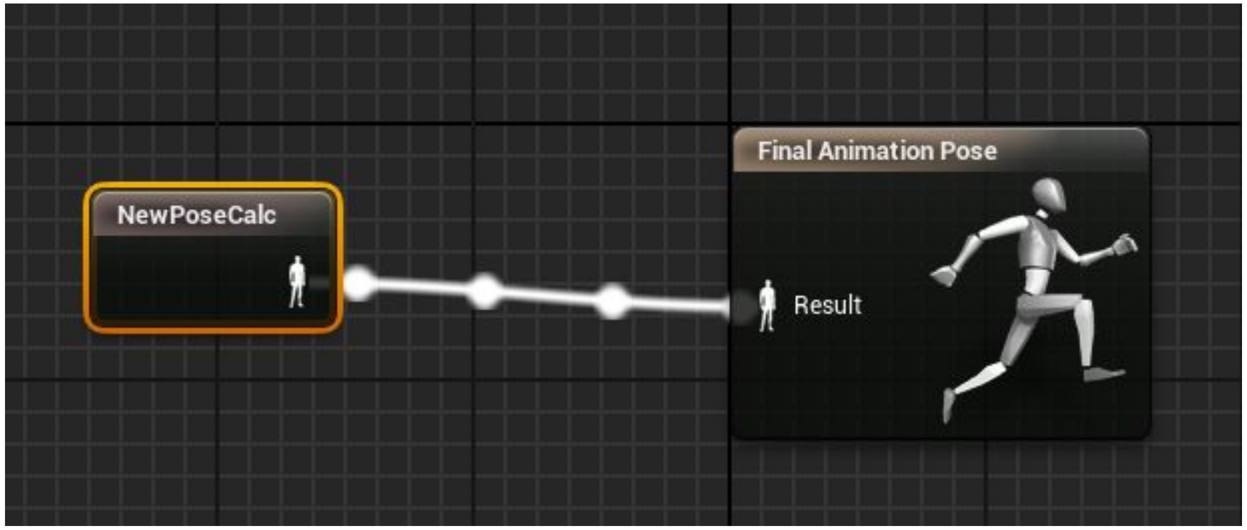
We can see the newly created **Avatar_Skeleton_AnimBlueprint** in ContentBrowser:



Double left-click on the Animation Blueprint to edit it. Right-click on **Anim Graph** to create a **NewPoseCalc** node and link it`s output to FinalAnimationPose node.

Of course you can also do some work on the output of **NewPoseCalc** node according to your requirement, and then output the final result to FinalAnimationPose node.

Compile **Anim Graph**:



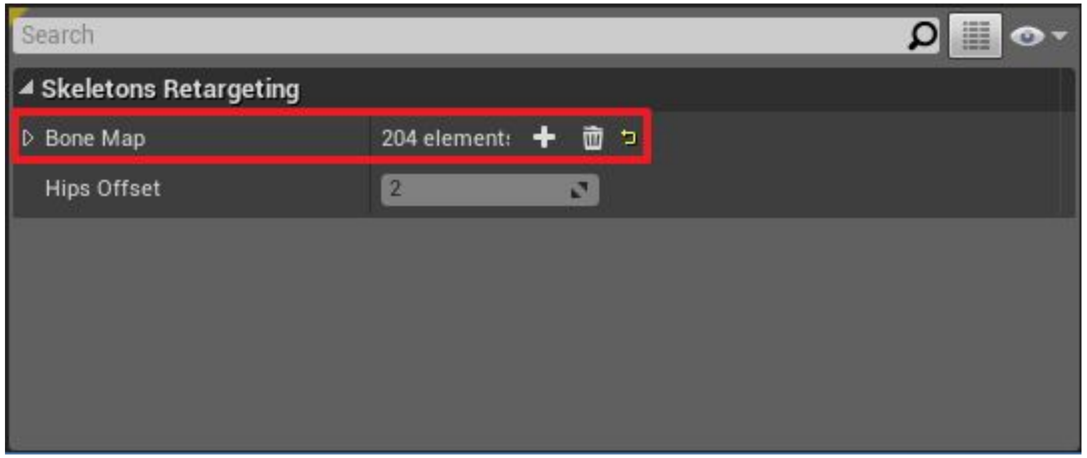
Skeleton Retargeting

PerceptionNeuron plugin defines a standard skeleton structure. The standard skeleton structure is made of 59 bones, they are:

```
Hips
RightUpLeg
RightLeg
RightFoot
LeftUpLeg
LeftLeg
LeftFoot
Spine
Spine1
Spine2
Spine3
Neck
Head
RightShoulder
RightArm
RightForeArm
RightHand
RightHandThumb1
RightHandThumb2
RightHandThumb3
RightInHandIndex
RightHandIndex1
RightHandIndex2
RightHandIndex3
RightInHandMiddle
RightHandMiddle1
RightHandMiddle2
RightHandMiddle3
```

```
RightInHandRing
RightHandRing1
RightHandRing2
RightHandRing3
RightInHandPinky
RightHandPinky1
RightHandPinky2
RightHandPinky3
LeftShoulder
LeftArm
LeftForeArm
LeftHand
LeftHandThumb1
LeftHandThumb2
LeftHandThumb3
LeftInHandIndex
LeftHandIndex1
LeftHandIndex2
LeftHandIndex3
LeftInHandMiddle
LeftHandMiddle1
LeftHandMiddle2
LeftHandMiddle3
LeftInHandRing
LeftHandRing1
LeftHandRing2
LeftHandRing3
LeftInHandPinky
LeftHandPinky1
LeftHandPinky2
LeftHandPinky3
```

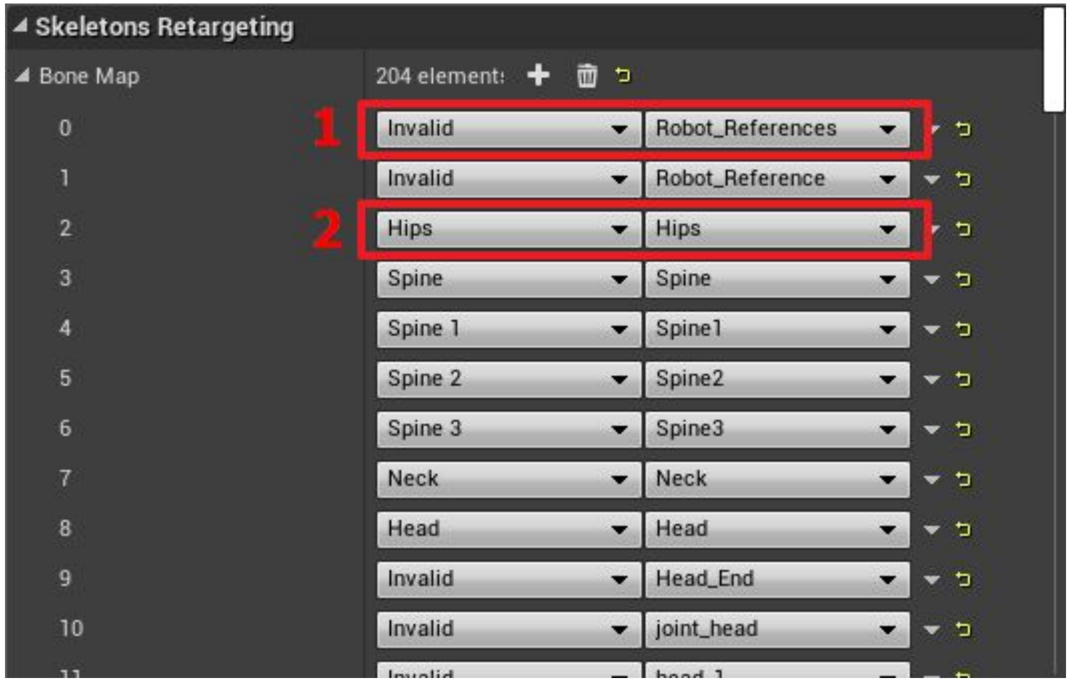
If the imported skeleton is not standard, you may need to do skeleton ragergeting to guarantee the character's movements driven by final motion capture data to reach your expectation. Select **NewPoseCalc** node in **Anim Graph**, it will display editing options about skeleton ragergeting in Details panel, as shown below:


Search

Skeletons Retargeting

Bone Map	204 element:
Hips Offset	2

We can expand **BoneMap** to edit SkeletonRagergeting:

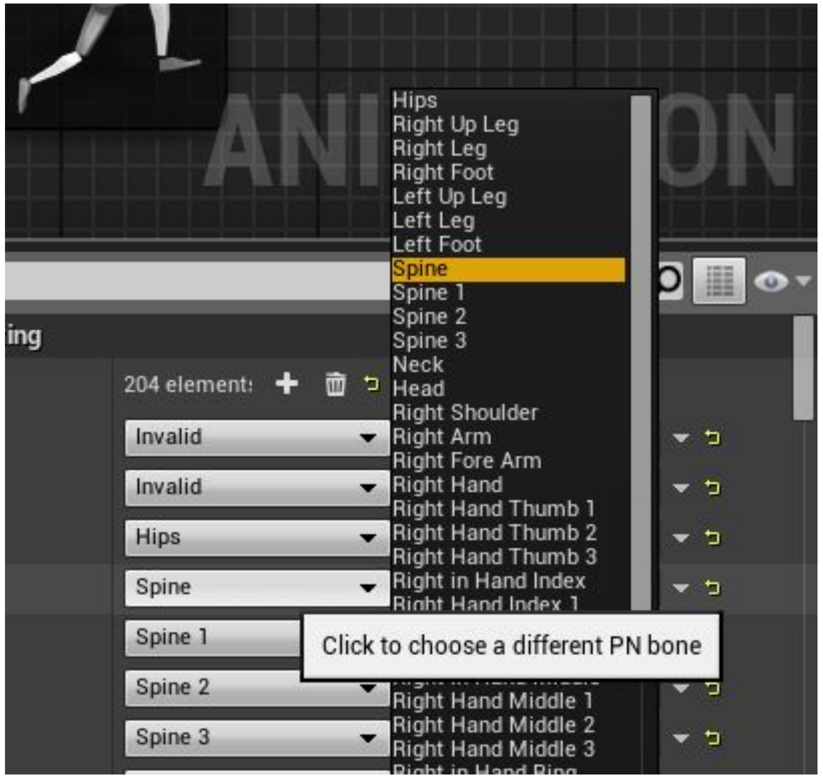

Skeletons Retargeting

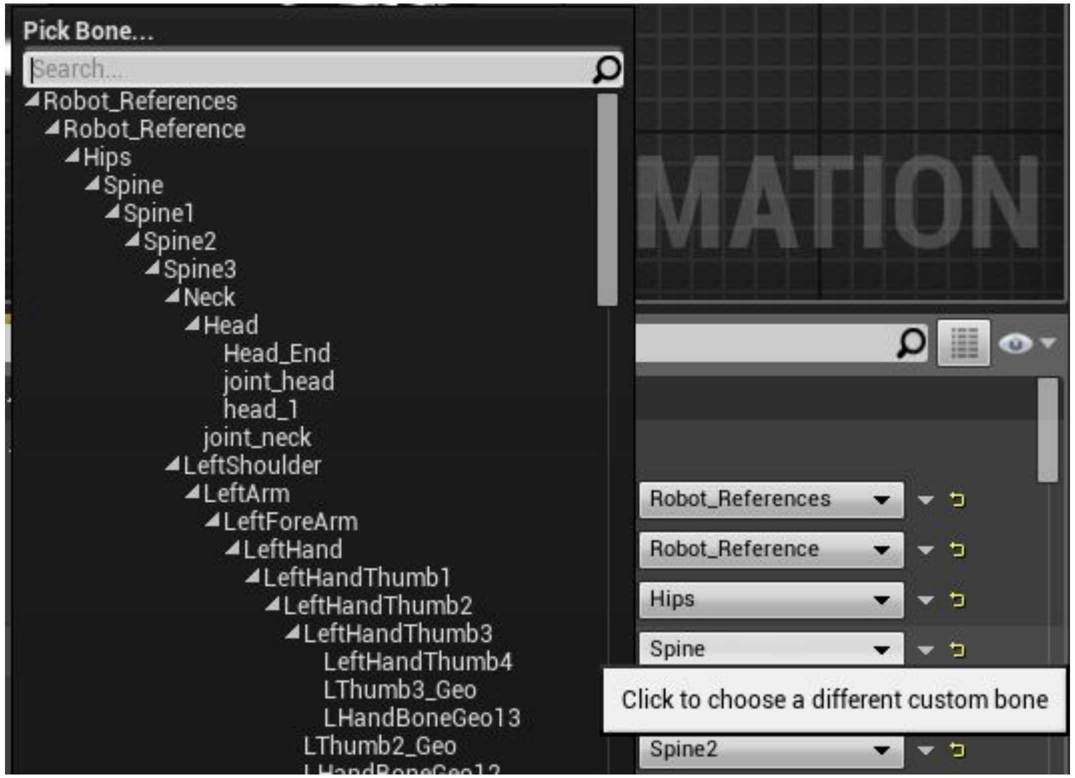
Bone Map 204 element:

0	Invalid	Robot_References
1	Invalid	Robot_Reference
2	Hips	Hips
3	Spine	Spine
4	Spine 1	Spine1
5	Spine 2	Spine2
6	Spine 3	Spine3
7	Neck	Neck
8	Head	Head
9	Invalid	Head_End
10	Invalid	joint_head

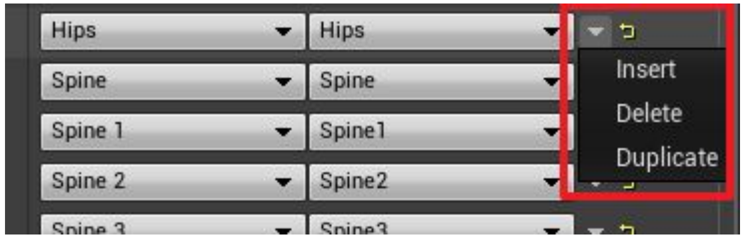
The left column are standard skeleton names defined by PerceptionNeuron, the right column are custom skeleton`s bone names. If the imported skeleton`s bone names completely fit the standard ones defined by PerceptionNeuron, then the two columns of skeleton names will automatically mapped correctly (pictured directly above, shown as 2). If the custom skeleton`s bone names contain the names which are not defined in the standard ones, then the plugin will automatically using **Invalid** to map the bone name (pictured directly above, shown as 1), and this bone will maintain relative still to the parent bone when animation is playing.

We can adjust the bone mapping relationship of skeleton ragergeting by clicking the bone names in the two columns:





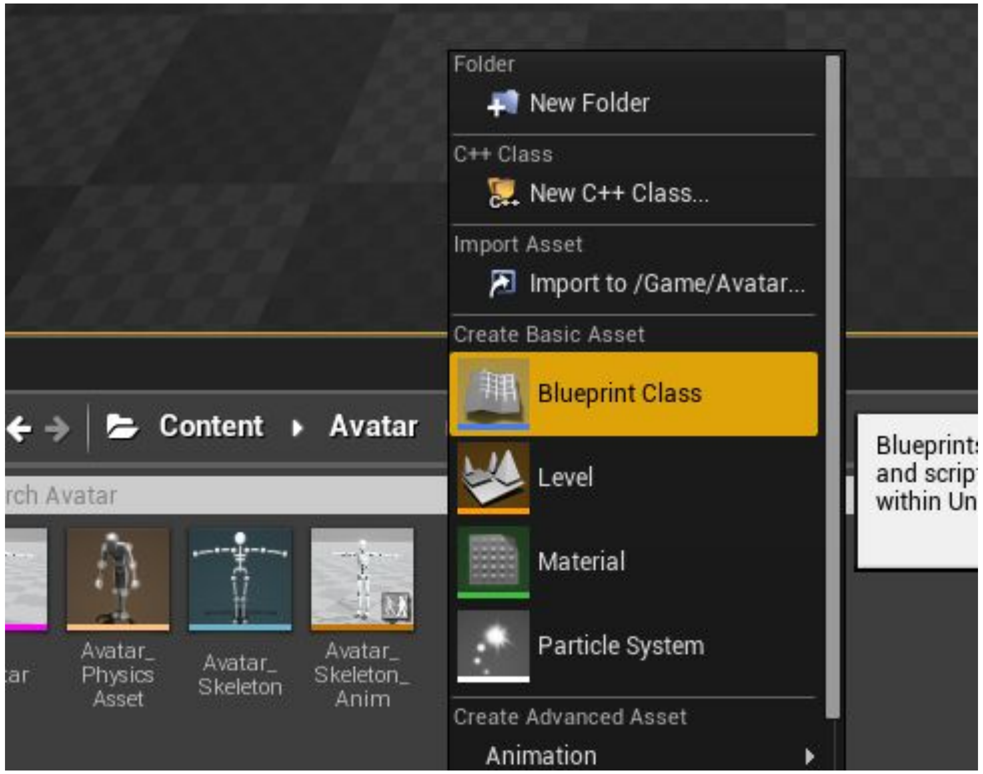
We can add and delete a couple of mapping of bone names:



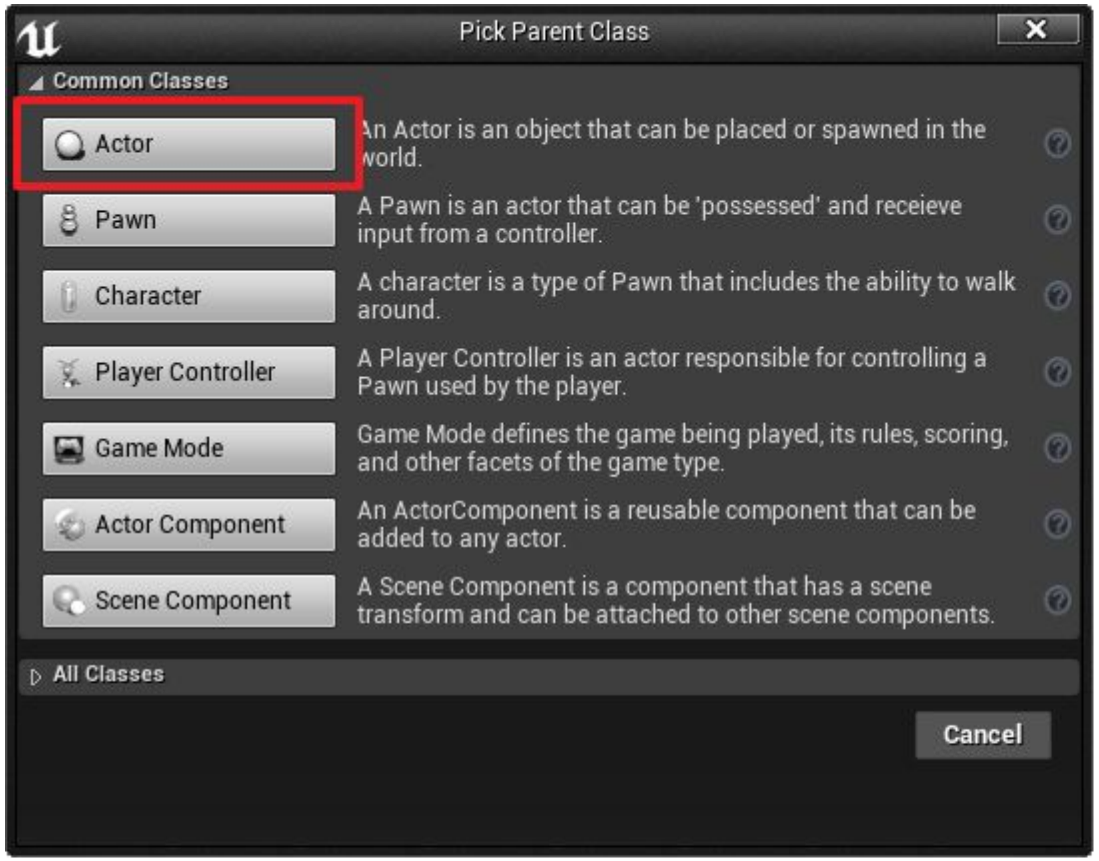
Note: It’s meaningless that any bone of skeleton is driven by multiple PN bones, so the names in the right column must be unique. The number of the bones in BoneMap may be more than actual imported one. This is because UE recognize mesh as skeleton, maybe it’s a bug. We can delete the bone mapping which name is invalid in the left column to improve performance when it is necessary.

Create Actor Blueprint

We create a Blueprint class and will use it to create test character:



For simplicity we choose **Actor** as **ParentClass,** create an actor Blueprint and named it as **"AvatarBlueprint"** :


Pick Parent Class

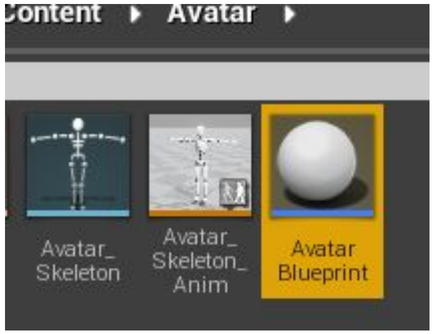
Common Classes

Actor	An Actor is an object that can be placed or spawned in the world.
Pawn	A Pawn is an actor that can be 'possessed' and receieve input from a controller.
Character	A character is a type of Pawn that includes the ability to walk around.
Player Controller	A Player Controller is an actor responsible for controlling a Pawn used by the player.
Game Mode	Game Mode defines the game being played, its rules, scoring, and other facets of the game type.
Actor Component	An ActorComponent is a reusable component that can be added to any actor.
Scene Component	A Scene Component is a component that has a scene transform and can be attached to other scene components.

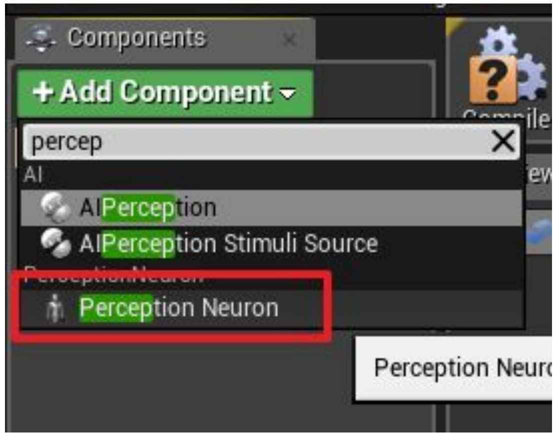
All Classes

Cancel

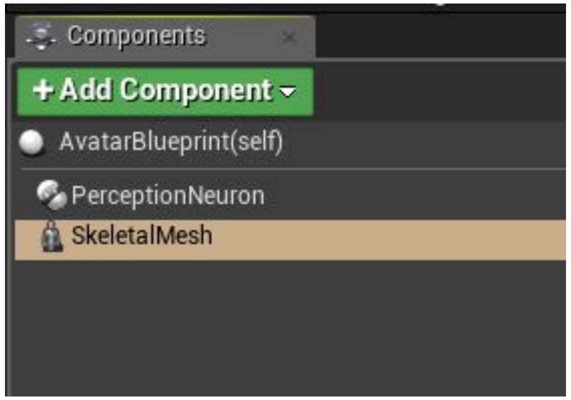
We can see the newly created **AvatarBlueprint** in ContentBrowser, double left-click on **AvatarBlueprint** to open the BP editor to edit **AvatarBlueprint**:



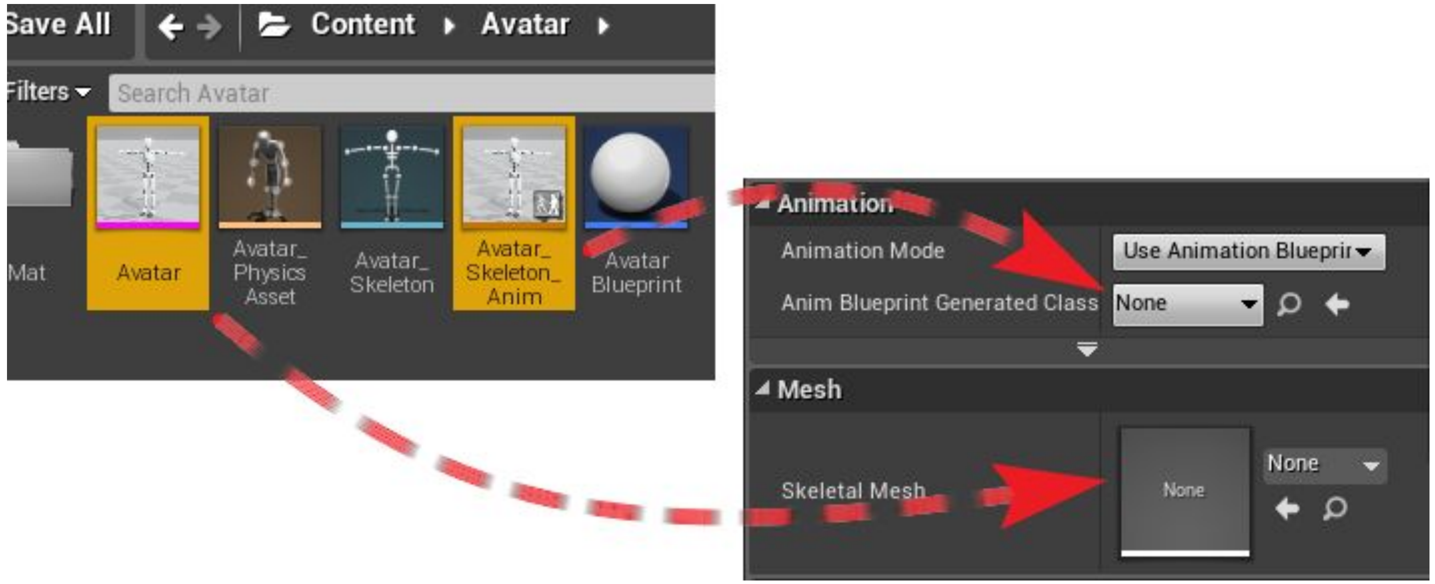
Click **AddComponent** to add **PerceptionNeuron** component:



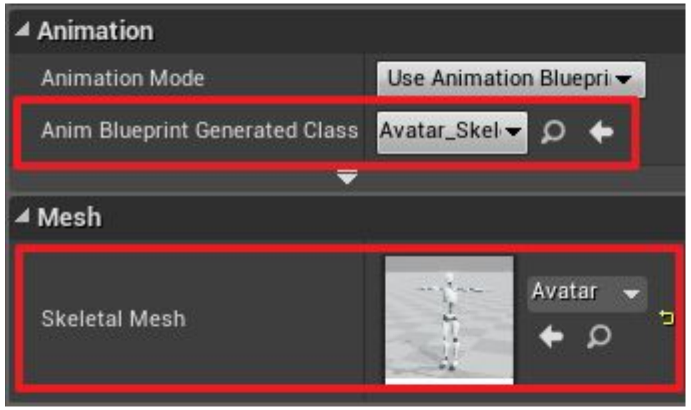
Click **AddComponent** again to add a **SkeletalMesh** component:



Click the newly created **SkeletalMesh** component, set **SkeletalMesh** component`s properties in the Details panel on the right side.



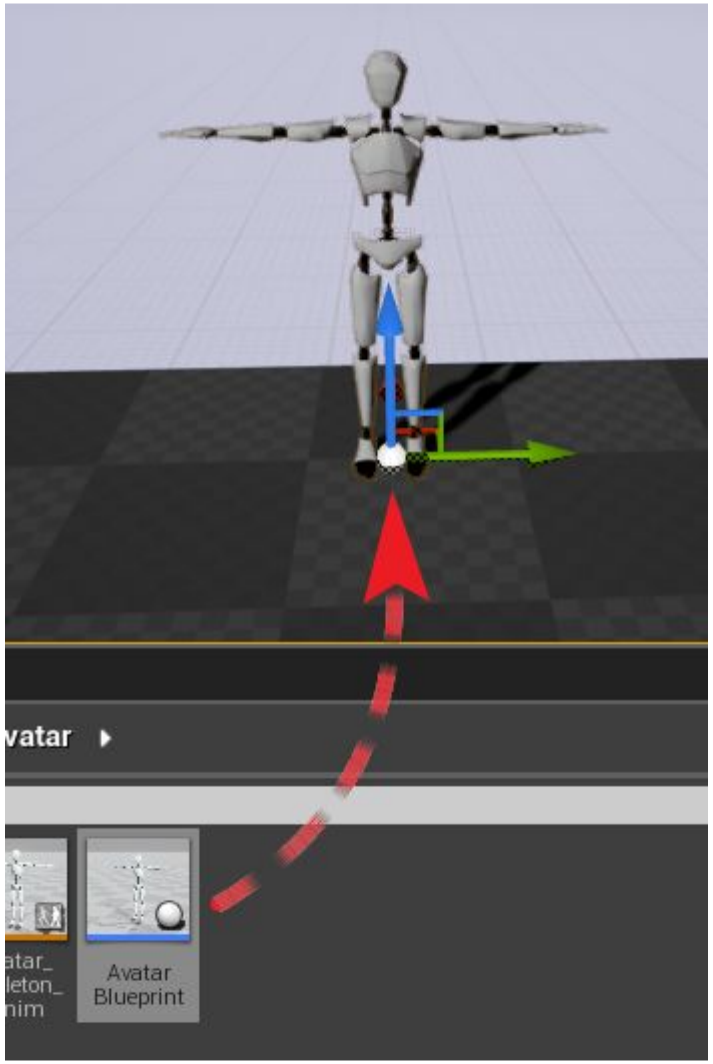
Here’s the result:



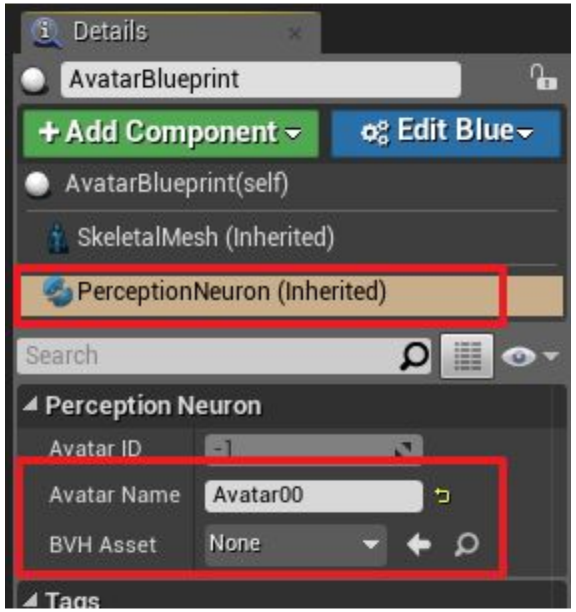
Compile **AvatarBlueprint** and close the BP editor.

Create Actor Instance and set up a data source

Drag and drop newly created **AvatarBlueprint** to the scene to create a new actor instance:



Select the newly created actor in the scene, then select **PerceptionNeuron** component under the actor in Details panel. We can find the properties **Avatar Name** and **BVH Asset** under **PerceptionNeuron** in Details panel:


Details

AvatarBlueprint

+Add Component | Edit Blue

AvatarBlueprint(self)

SkeletalMesh (Inherited)

PerceptionNeuron (Inherited)

Search

Perception Neuron

Avatar ID	-1
Avatar Name	Avatar00
BVH Asset	None

Tags

Attribute name	Description
Avater Name	Avater Name is used for identify network data source stream Name used by an actor when one actor uses network data source to drive animation. (Each avatar`s motion capture data send by Axis Neuron is identified by a Avatar Name, we can use the Avatar Name to match an actor with it`s data source stream). Network data source will be disabled when the property BVH Asset is assigned and the actor animation will be driven by BVH asset data source.
BVH Asset	BVH Asset property is a reference of BVH asset. To create BVH asset we need to use BVH motion capture data. Network data source will be disabled when the property BVH Asset is assigned and the actor animation will be driven by BVH asset data source.

BVH asset data source

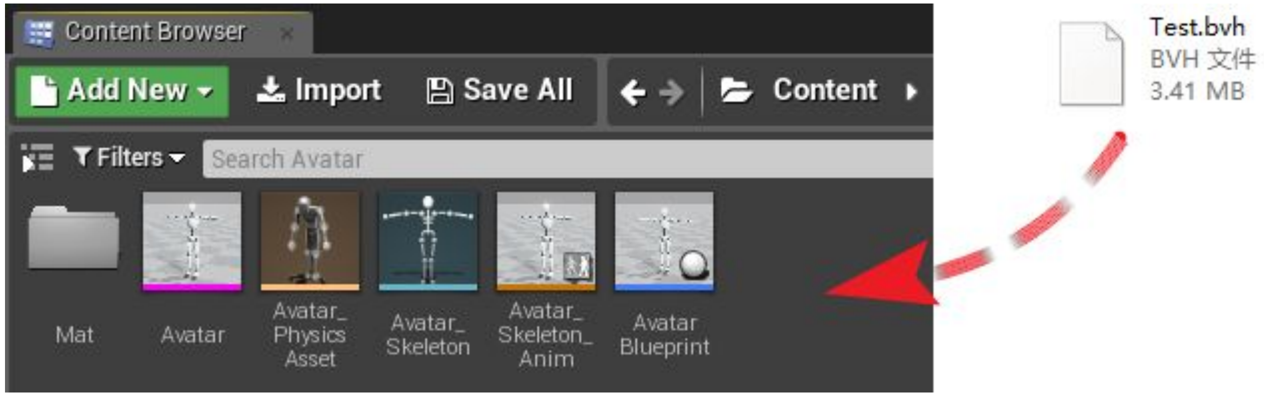
BVH(BiovisionHierarchy) is a file format used for storing motion capture data which is developed by Biovision. We can import BVH format data file to UnrealEngine to generate BVH asset. The motion capture data saved in BVH file will be stored in BVH asset. Set BVH asset to an actor as data source, then the actor will automatically read the motion capture data from it to drive animation.

Import BVHfile

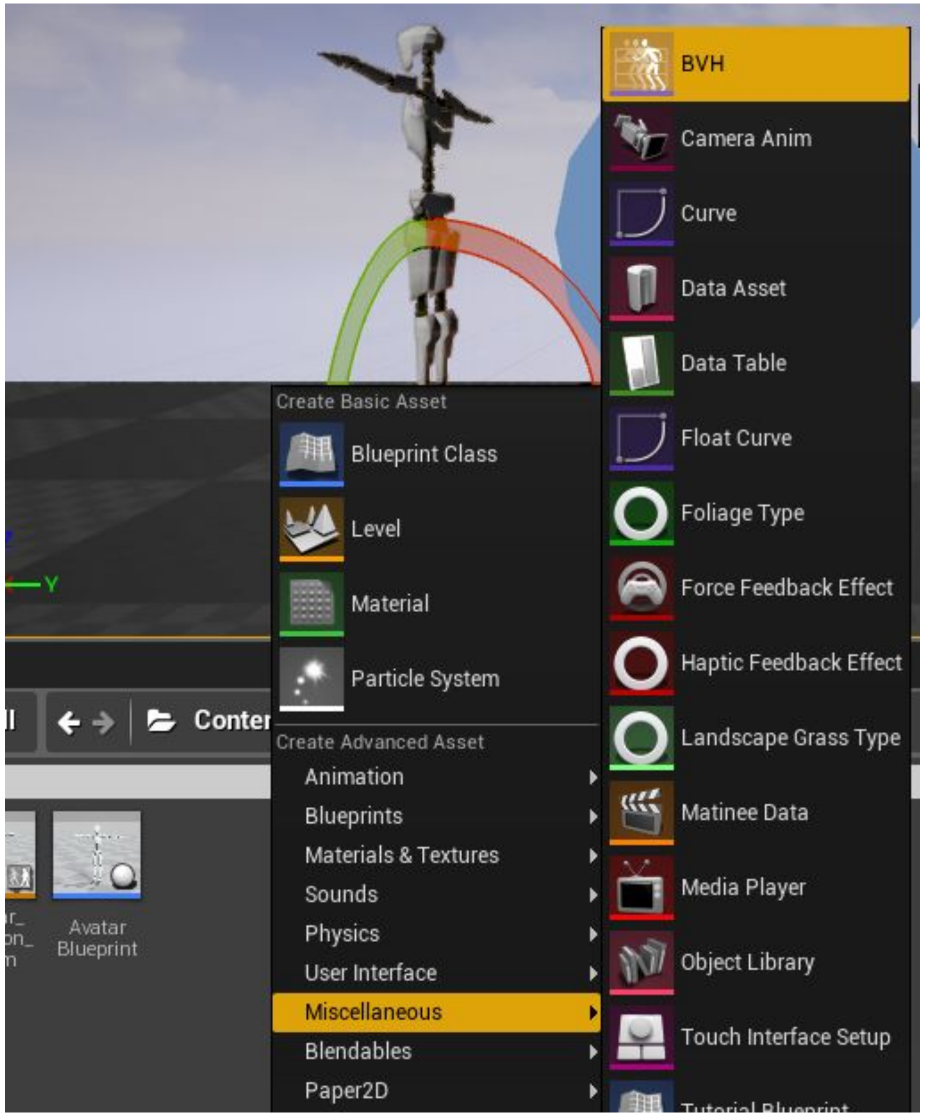
We can import BVH format data file via 3 ways:

1. Put BVH format data file in content folder of project directory, then UnrealEngine will automatically detect file changes and import BVH format data file to generate BVH asset。

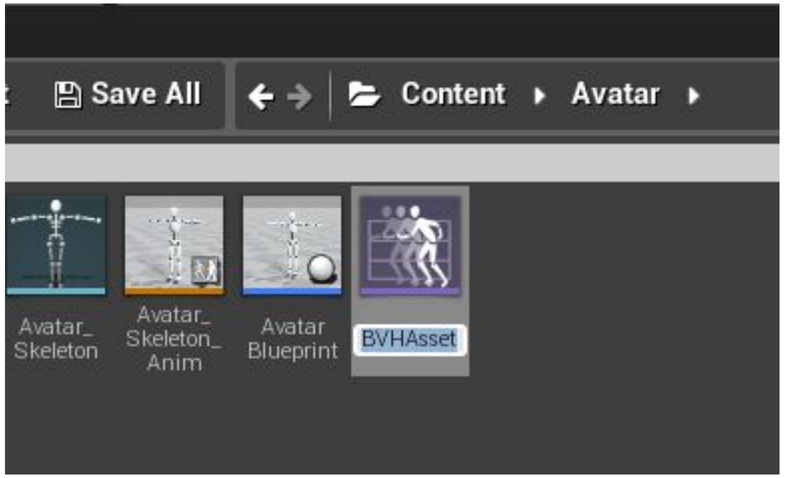
2. Drag BVH format data file to **ContentBrowser** panel:



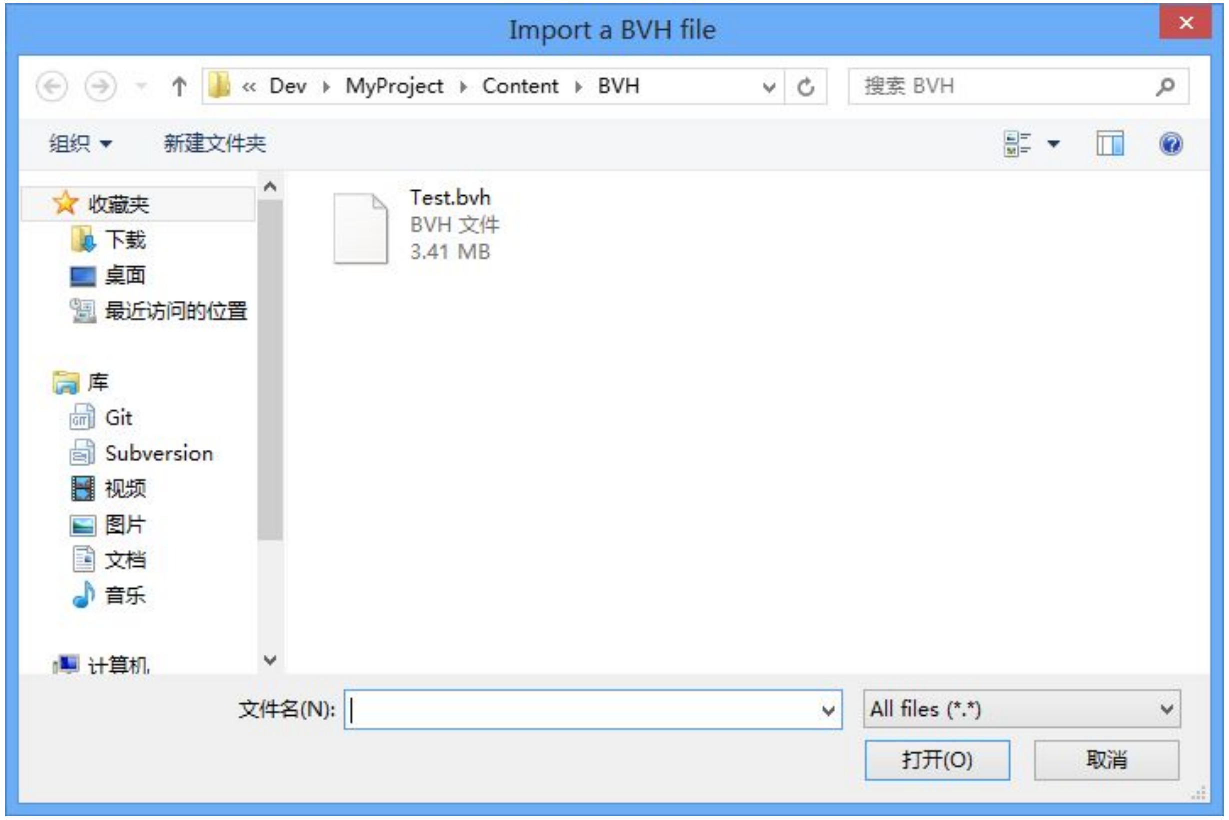
3. Right-click on the blank of **ContentBrowser** panel, select **Miscellaneous -> BVH** in the right-click menu:



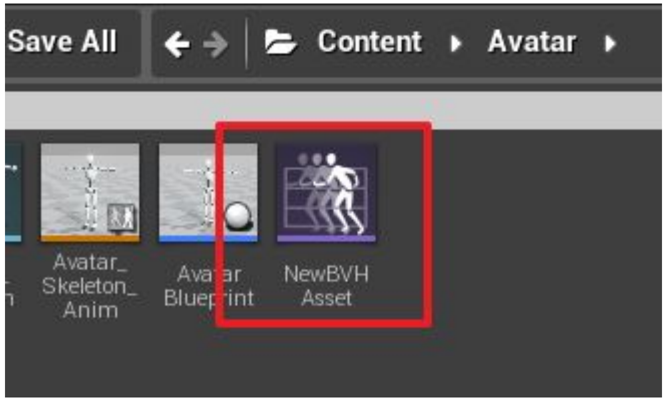
We can specify an asset name or use the default one:



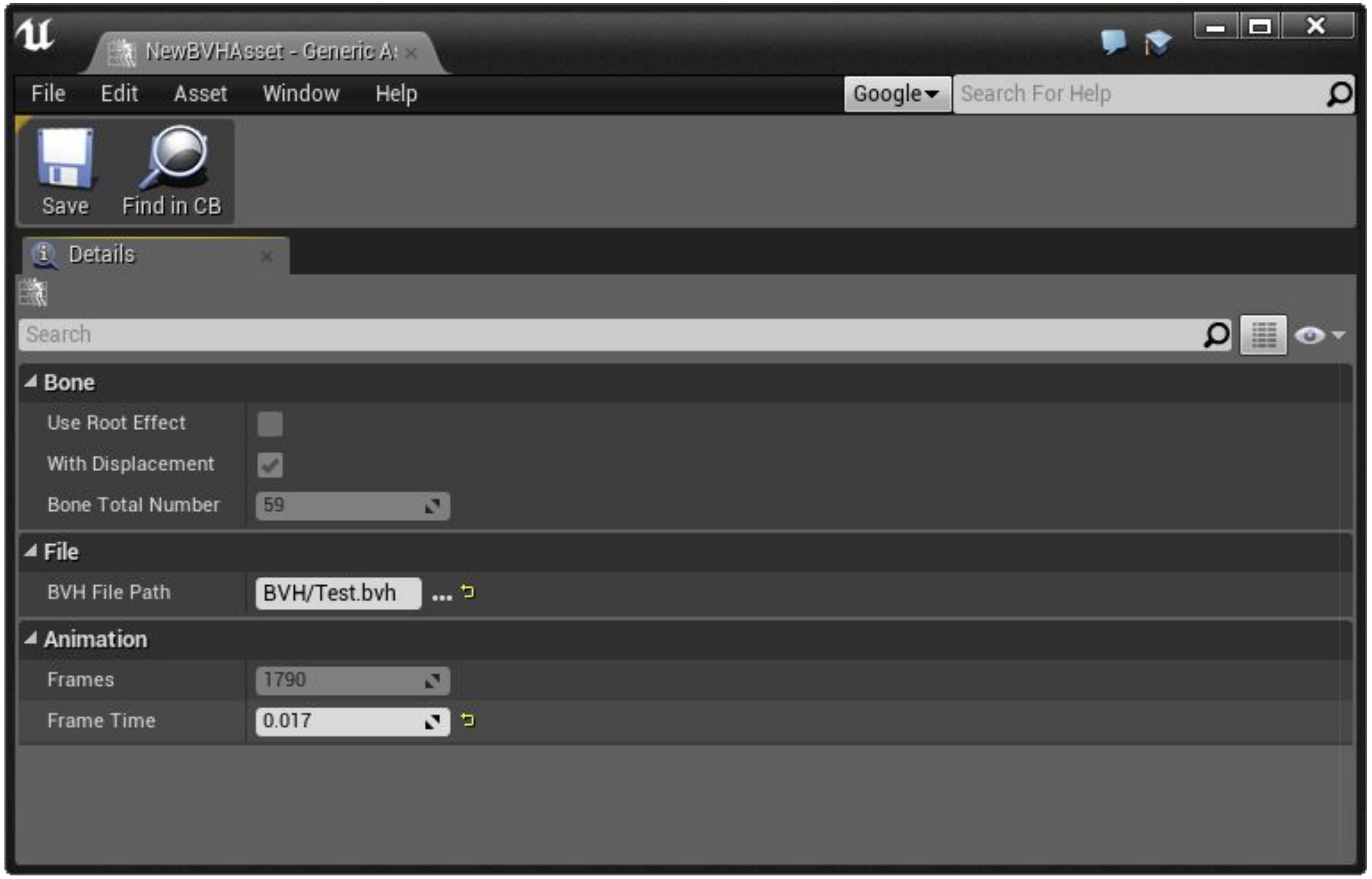
Left-click and it will pop up a dialog to let us choose a BVH data file to be imported.



Choose **Test.bvh** and import:



We can double click on BVH asset to open the editor to check the properties of BVH asset and edit it:



Bone

Use Root Effect	☐
With Displacement	✓
Bone Total Number	59

File

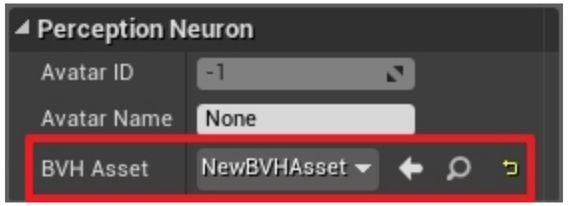
BVH File Path	BVH/Test.bvh

Animation

Frames	1790
Frame Time	0.017

Use BVH asset

Now BVH asset has been created successfully. We can set the BVH asset to the data source of an actor:

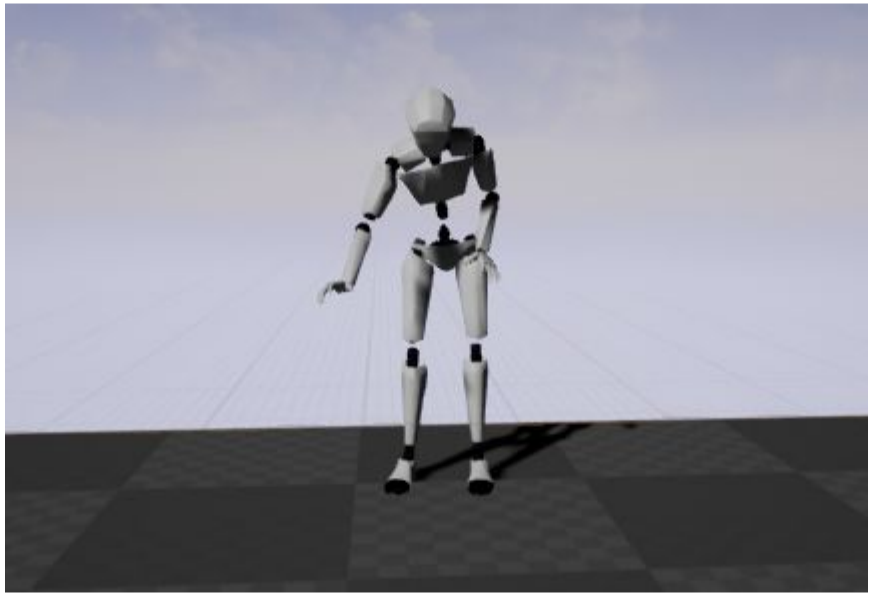

Perception Neuron

Avatar ID -1

Avatar Name None

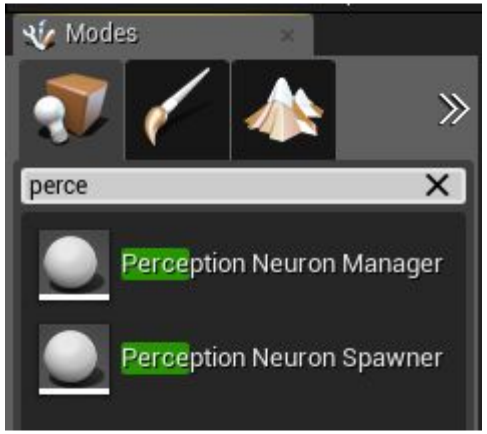
BVH Asset NewBVHAsset

Now the test actor is ready to play motion capture data:



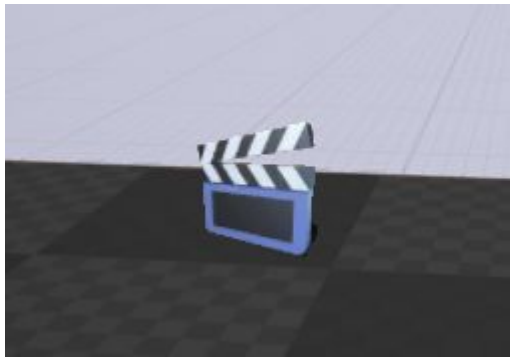
Network data source

We should create **Perception Neuron Manager** and **Perception Neuron Spawner** before using network data source：

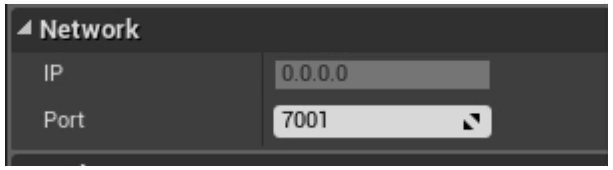


Perception Neuron Manager is used for creating the network communication environment and disposing received motion capture data. **Perception Neuron Spawner** contains some properties for creating an actor, such as class type, initial born position, network data stream name and so on.

First, we put a **Perception Neuron Manager** into the scene:



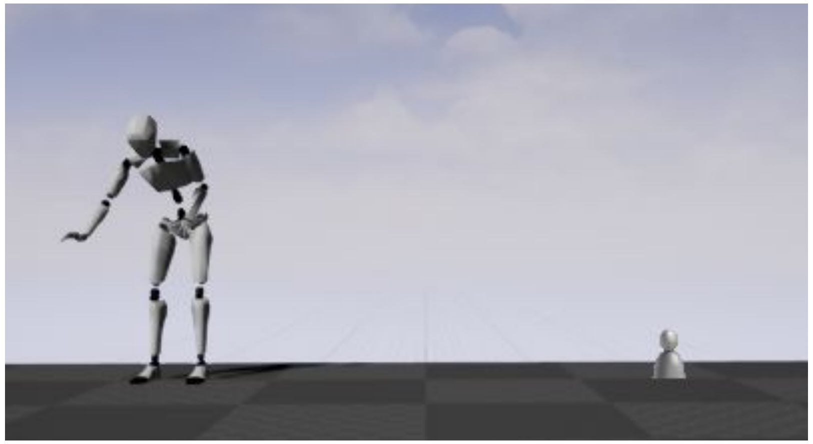
Set network properties for **Perception Neuron Manager**.IP can`t be edited,plugin can get the host IP automatically and show it in Details panel;Set the port number with the port in Axis Neuron settings.



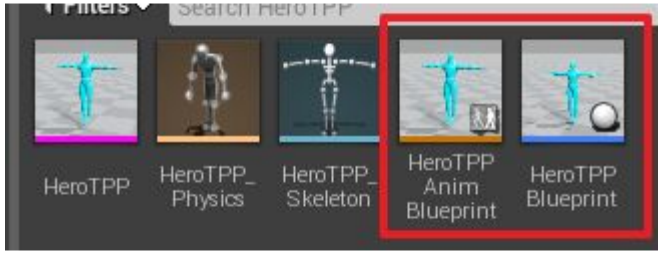
Network

IP	0.0.0.0
Port	7001

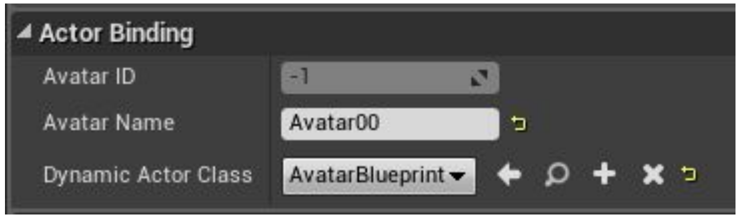
Create a **Perception Neuron Spawner** in the scene to indicate the initial born position for the actor:



I use **HeroTPP** to make **HeroTPPBlueprint** and **HeroTPPAnimBlueprint**, and have deal with the skeleton retargeting:



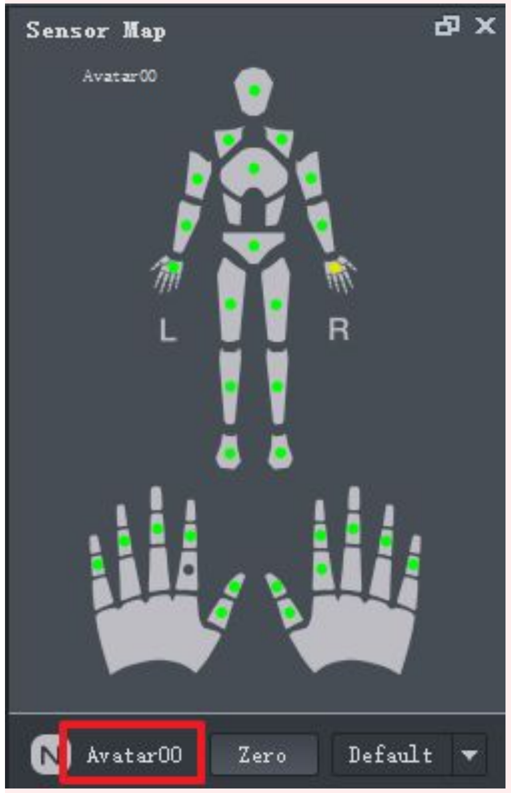
Set the newly created **HeroTPPBlueprint** to spawner and set the receiving network data stream name with **Avatar00** for spawner:



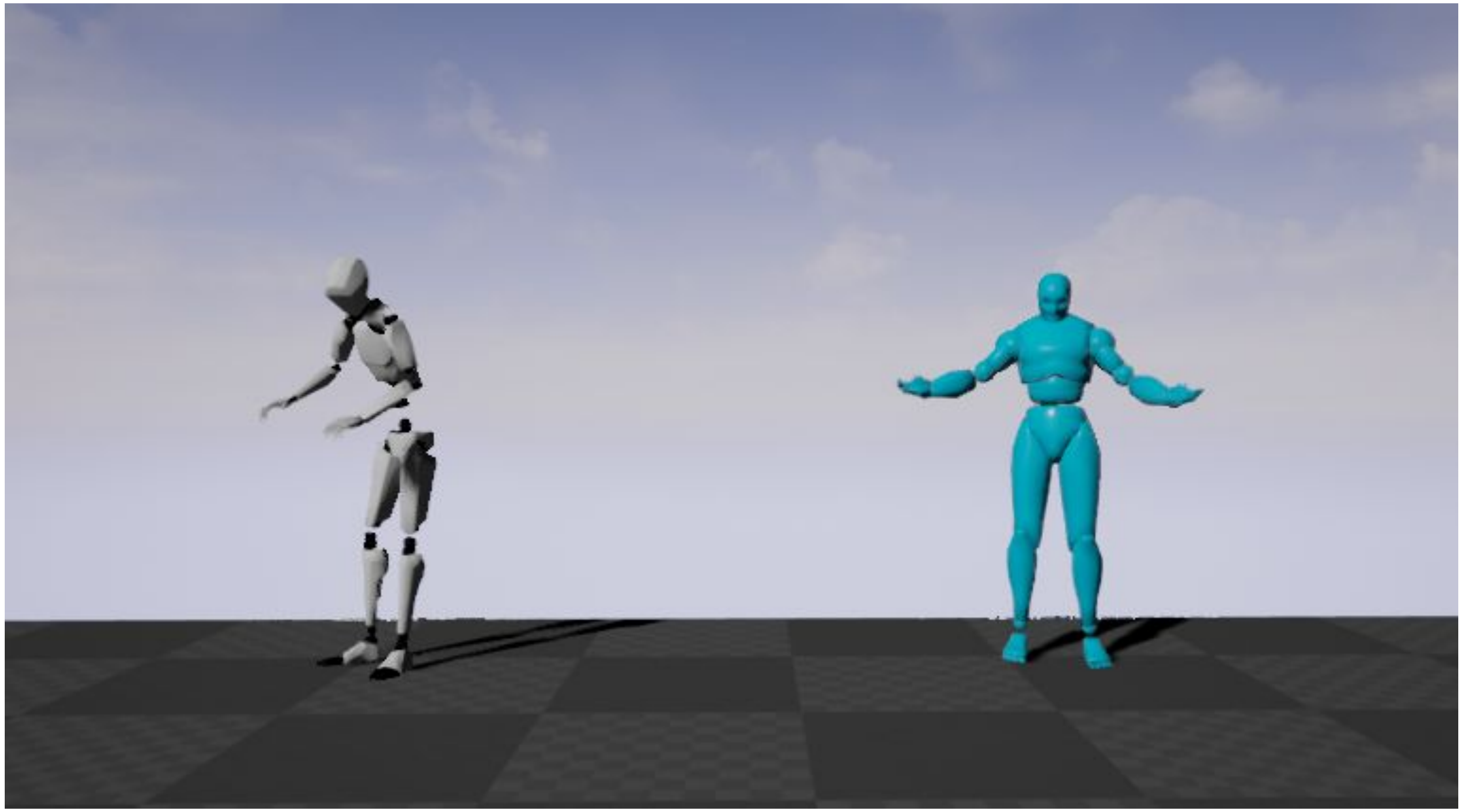
Actor Binding

Avatar ID	-1
Avatar Name	Avatar00
Dynamic Actor Class	AvatarBlueprint

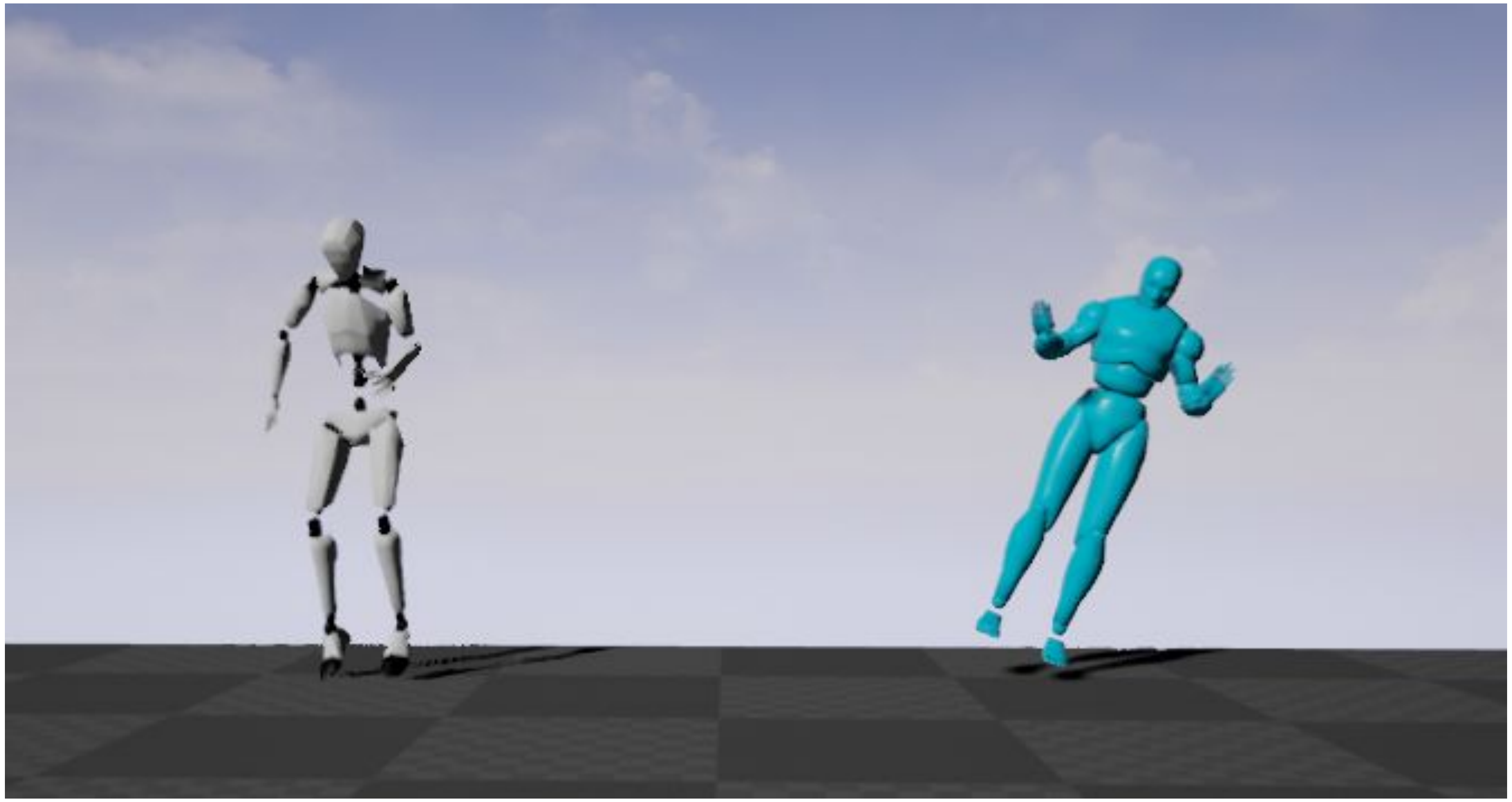
Note:The default network data stream name for Axis Neuron is Avatar00. The default value is used here, but the name can be modified in Neuron Axis:



Click **Play** and run Axis Neuron to send motion capture data, we can see that a new HeroTPP actor has been created dynamically in the scene and started to play animation:



We can rotate the direction of spawner to run different results:



Note: Axis Neuron provides TCP and UDP two kinds of protocol to send network data, but PerceptionNeuron plugin only supports UDP protocol to receive network data currently.

Support

PerceptionNeuron plugin is compatible with UnrealEngine4.9~4.11 and all of the platforms supported by UnrealEngine4.

You can get the test DEMO used in this document through https://104.156.239.211/svn/PerceptionNeuronPluginForRelease/Demo.

Please contact me if you have any problem with PerceptionNeuron plugin, my e-mail is weiboreal@gmail.com.